Роль нейронных сетей в программировании и разработке веб-приложений

The Role of Neural Networks in Programming and Web Application Development

Башаров Айдар Рустемович

Студент 1 курса, бэкенд-разработчик

Институт среднего профессионального образования

Уфимский университет науки и технологий, Уфимский авиационный техникум

Россия, Республика Башкортостан, Уфа, ул. Ленина, 61

e-mail: viserd.yt@gmail.com

Basharov Aydar Rustemovich

Student 1 term, backend-developer

Institute of Secondary Vocational Education

Ufa University of Science and Technology, Ufa Aviation Technical School

Russia, Republic of Bashkortostan, Ufa, st. Lenina, 61

e-mail: viserd.yt@gmail.com

Аннотация

В статье подробно описано, как влияют нейросети на разработку веб-приложений, из методов исследования были взяты конкретные примеры для настоящих проектов разного уровня, в итоге установлено, что нейросети плохо генерируют код для сложный приложений и немалый функционалом, в свою очередь хорошо справляются с рутинными и легкими задачами, которые не требуют запоминания многих зависимостей и другой информации.

Annotation

The article describes in detail how neural networks influence web application development. Specific examples for real projects of various scales were taken from the research methods. As a result, it was established that neural networks are poor at generating code for complex applications with considerable functionality, while they cope well with routine and easy tasks that do not require remembering many dependencies and other information.

Ключевые слова: Нейронные сети, роль нейронных сетей, программирование, роль нейронных сетей в разработке веб-приложений, веб-приложения, веб-разработка.

Key words: Neural networks, the role of neural networks, programming, the role of neural networks in web application development, web applications, web development.

Введение:

Нейро́нные сети сегодня в наше время нейросети используются повседневно, наверняка каждый человек когда-либо использовал **нейросеть** или ИИ (Искусственный интеллект) для решения какой-либо задачи. Конкретно в веб-разработке нейросети используются для решения разных задач: от рутинного кода[1] до создания целый сайтов, из-за чего появилось целое движение программистов, которые называют себя “вайб кодеры” — программисты, которые используют только ИИ и не притрагиваются сами к коду[1].

Движений **вайб кодинга** стало опасным для **веб-разработки**, ввиду игнорирования проверки кода, написанного ИИ, который может содержать ошибки и угрозы для сайта[2].

Цель:

Провести **сравнительный анализ** качества генерации кода на языке TypeScript тремя нейросетевыми моделями (**Алиса**, **DeepSeek**, **ChatGPT**) для оценки их готовности к использованию в промышленной разработке.

Задачи:

1. **Разработать методику сравнительного анализа** качества кода, сгенерированного нейросетевыми моделями, включающую:
    a. Критерии оценки типизации (соответствие стандартам TypeScript, использование any-типов, полнота аннотаций).
    b. Критерии оценки логики (читаемость, отсутствие избыточных конструкций, корректность алгоритмов).
    c. Критерии оценки соответствия техническому заданию
2. **Провести экспериментальное исследование** по генерации кода для трех типовых задач:
    a. Реализация математических алгоритмов (квадратные уравнения).
    b. Создание служебных классов (система логирования).
    c. Разработка UI-компонентов (React-компоненты).
3. **Выявить и классифицировать типичные ошибки**, допускаемые нейросетевыми моделями при генерации TypeScript-кода:
    a. Ошибки типизации (количество, виды, критичность).
    b. Логические ошибки (архитектурные и алгоритмические).
    c. Ошибки соответствия техническому заданию.
4. **Проанализировать различия в подходах** к генерации кода между исследуемыми нейросетевыми моделями:

---

[1] Рутинный код — повторяющийся код, не требующий особо обдумывания задачи.

    a. Сравнительный анализ по количеству и типам ошибок.
    b. Оценка сложности и поддерживаемости генерируемого кода.
    c. Анализ стилевых особенностей генерируемого кода.
5. **Разработать практические рекомендации** для разработчиков по:
    a. Выбору нейросетевой модели для различных типов задач.
    b. Методике проверки и доработки сгенерированного кода.
    c. Оптимизации промптов для минимизации ошибок.
6. **Сформулировать требования к улучшению** нейросетевых моделей для генерации кода:
    a. Приоритетные направления для дообучения моделей.
    b. Рекомендации по обработке типичных сценариев ошибок.
    c. Предложения по интеграции статических анализаторов кода.
7. **Выявить зависимость качества** написания промта.
    a. Насколько хорошо нужно прописывать техническое задание для нейросети.
    b. От чего зависит качество написанного технического задания.

Научная новизна заключается в создании комплексной методики оценки качества генерации TypeScript-кода и проведении сравнительного анализа современных нейросетевых моделей на репрезентативной выборке практических задач.

Практическая значимость состоит в разработке конкретных рекомендаций для разработчиков по эффективному использованию нейросетевых ассистентов в реальных проектах на TypeScript.

Теоретическая часть:

Словарь:

- **Код** — текст программы, написанный на языке определенном языке программирования.
- **Комментарий** — пояснительная часть кода, которая не используются в программе.
- **Типизация**[3] — система правил для работы с данными[4].
- **Функция** — часть кода, которую можно выполнять из любой части программы, задавая ей различные данные на вход безграничное количество раз.
- **Класс** — модель для создания объектов определенного типа, описывающая их структуру (набор полей и их начальное состояние) и определяющая алгоритмы (функции или методы) для работы с этими объектами[5].
- **Методы** — функция, служащая определенному классу в коде.
- **Массив** — набор элементов в списке, содержащие один тип данных.

- **Константа** — неизменяемое значение.

Описание:

Нейросети стали **важным инструментом** в программировании. Они могут значительно **упростить** процесс написания кода, **автоматизируя** рутинные задачи и предлагая решения проблем. Это позволяет разработчикам сосредоточиться на более сложных аспектах проектирования и архитектуры программного обеспечения.

Помимо этого нейросети способны **уменьшите количество** рутинного написания кода у программистов, что **позволяет** программистам лучше следить за кодом и **уделять больше времени** более сложным и фундаментальным конструкциям. Нейросети способны генерировать целые функции, методы и даже классы за считанные секунды.

Мы возьмем три нейросети: Алиса, DeepSeek и ChatGPT для генерации кода разного уровня от легких функций до проектов уровня бизнес-приложения. Каждой нейросети мы дадим одинаковые промты[2] для генерации коде. Также у каждой нейросети будет доступ к Интернету:

1. Создай функцию для решения квадратного уравнения через дискриминант на языке программирования TypeScript.
2. Напиши класс, которые будет иметь надстройку над консолью, выполняющий функцию логирования сообщений в консоль. класс должен иметь методы чтения и вывода сообщений в консоль. Пиши на языке программирования TypeScript.

С этого момента мы будет давать нейросетям более подробные промты, чтобы они писали более хороший код, так как далее идут более сложные задачи:

3. Разработай два компонента на следующем стеке: React + Tailwind CSS + TypeScript: Dropdown и Modal.
    - 3.1. Dropdown должен при нажатию вне компонента или по кнопке Esc.
    - 3.2. Modal меню должно закрываться по нажатию на кнопку закрытия или по кнопке Esc.
    - 3.3. Выноси логику (хуки, функции, константы и дочерние компоненты) больших ("главных", родительских) компонентов на дочерние файлы или функции.
    - 3.4. В исходном коде у тебя должно получится минимум 2 файла: dropdown.tsx и modal.tsx.

[2] Промт — запрос/инструкция для нейросети.

3.5. Можешь использовать react-icons и другие библиотеки для иконок.
3.6. Используй преимущественно React-хуки или функции, совместимые с ними.
3.7. Старайся не комментировать код — избавляйся от документирования и комментирования того, что пишешь. Только код. (пути к файлам в начале кода писать разрешено).

Также мы будем игнорировать все комментарии (кроме jsdocs[8]). После всех анализов, мы напишем свой код, убирая все ошибки их кода нейросетей.

Практическая часть:

Задание номер один: “Решение квадратного уравнение через дискриминант на языке программирования TypeScript.”

Ошибки в коде нейросети Алисы (код 1):

1. Плохая типизация в коде.
    a. В последнем условии функция возвращает[6] массив строк, что служит ошибки типизации, так как у функции четко написано, что она возвращает массив чисел или строку (строки: 32-33 и 8).
    b. Функция возвращает не определенный массив чисел (то есть нет конкретного количества элементов в массиве) (строки: 22, 26 и 32-33).
2. Плохая логика в коде.
    a. Функция может вернуть строку, содержащую ошибку, что только добавляет работы разработчику, работающему с этой функцией.
    b. Лишние `else-if` конструкции[7]. Они не требуются, так как после того, как будет выполнено одно из этих условий, функция вернёт значения, а значит код не продолжит свое выполнение.

Ошибки в коде нейросети DeepSeek (код 2):

1. Плохая типизация в коде. Функция возвращает не определенный массив чисел (строки: 1, 5, 7, 14, 17, 24).

2. Плохая логика в коде:
    a. Вложенные условные конструкции[7], которые ухудшают чтение и понимание кода (строки: 3-8).
    b. Возможность решение линейного уравнения: при коэффициенте a равным нулю мы должны кидать ошибку о переходе уравнения в линейным вид (строки: 3-8).
    c. Нет выноса выражений в отдельные константы (строки: 5, 7, 14, 17).

Ошибки в коде нейросети ChatGPT (код 3):

1. Плохая типизация в коде. Функция возвращает не определенный массив чисел (строки: 9, 12, 20, 23, 36).
2. Плохая логика в коде:
    a. Лишние `else-if` конструкции. Они не требуются, так как после того, как будет выполнено одно из этих условий, функция вернёт значения, а значит код дальше не продолжит идти (строки: 17-27).
    b. Возможность решение линейного уравнения: при коэффициенте a равным нулю мы должны кидать ошибку о переходе уравнения в линейным вид (строки: 10-13).

Задание номер два: “Класс для чтения и вывода сообщений в консоль на языке программирования TypeScript.”

Ошибки в коде нейросети Алиса (код 4):

1. Плохая типизация в коде:
    a. Импорт `readline` через `require` не определяет ему тип, из-за чего input в строке 48 имеет неопределенный тип.
    b. `error` в строках 70-72 имеет неопределенные тип, из-за чего нельзя точно знать, имеется ли `message` у него.
2. Плохая логика в коде:
    a. Желательно добавить возвращаемые значения у методов (строки: 12-14, 17-19, 22-24, 27-29, 32-38).
    b. Изменяемые значения в константе `AliceConsoleLogger.COLORS` (строки: 3-9).

c. Строковые литералы в параметрах метода (строки: 13, 18, 23, 28).
d. Отсутствие вынесения логики за методы (строки: 34-37, 43-51).

Ошибки в коде нейросети DeepSeek (код 5):

1. Плохая логика в коде:
   a. Использование `isEnabled` и `enabled` (два разные формата написания булевых значений) в коде (строки 3, 7, 32-33).
   b. Наличие строковых литералов `[WARN]`, `[ERROR]`, `[INFO]` (строки: 38, 42, 46).
   c. Желательно добавить возвращаемые значения у методов (строки: 6-14, 27-29, 32-34, 37-39, 41-43, 45-47).
2. Плохо выполненное техническое задание: Отсутствие чтения данных с консоли.

Ошибки в коде нейросети ChatGPT (код 6):

1. Плохая типизация в коде:
   a. Поле называется `levelsOrder`, когда тип этого поля `LogLevel[]` (строка 12).
   b. Использование `any` типа (строки: 7, 51, 55, 59, 63).
   c. Отсутствие типов возвращаемых значений у методов (строки: 17, 32, 51, 55, 59, 63).
2. Плохое логика в коде:
   a. Создание массива на основе типа, а не наоборот (строки: 1, 12).
   b. Использование лишних операций (строки: 26, 36-40).
   c. Отсутствие выноса логики (строки: 34-35).
   d. Нагруженность (излишняя сложность) в коде.
   e. Плохое наименование в коде (строки: 12, 17, 22, 32, 46).
   f. Желательно добавить возвращаемые значения у методов (строки: 17-29, 32-43, 51-53, 55-57, 59-61, 63-65).
3. Плохо выполненное техническое задание: Отсутствие чтения данных с консоли.

Задание номер три: “Интерактивные компоненты”

Ошибки в коде нейросети Алиса (код 7, 8): dropdown.tsx, modal.tsx:

1. (`dropdown.tsx`) Плохая типизация в коде:
    a. Использование утверждения типов (`as Node`) (строка 15).
    b. Отсутствие желательного `| null` в `useRef` (строка 12).
2. (`dropdown.tsx`) Плохая логика в коде: Использование магических строк (строка 21).
3. (`modal.tsx`) Плохая типизация в коде: Отсутствие желательного `| null` в `useRef` (строка 12).
4. (`modal.tsx`) Плохая логика в коде: Неиспользуемые импорт `useState` (строка 1).

Ошибки в коде нейросети DeepSeek (код 9, 10): dropdown.tsx, modal.tsx:

1. (`dropdown.tsx`) Плохая типизация в коде:
    a. Отсутствие желательного `| null` в `useRef` (строка 11).
    b. Использование утверждения типов (`as Node`) (строка 23).
    c. Отсутствие импорта `as type` (строка 1, 49).
2. (`dropdown.tsx`) Плохая логика в коде:
    a. Использование магических строк (строка 15).
    b. Использование вложенных в строку условий (строка 38).
    c. Использование оператора опциональной поддержки (строка 23).
3. (`modal.tsx`) Плохая типизация в коде:
    a. Отсутствие желательного `| null` в `useRef` (строка 11).
    b. Использование утверждения типов (`as Node`) (строка 23)
4. (`modal.tsx`) Плохая логика в коде: Использование длинных условий (строка 23).

Ошибки в коде нейросети ChatGPT (код 11-14): components/dropdown.tsx, components/modal.tsx, hooks/useOutsideClick.ts, App.tsx:

1. (`hooks/useOutsudeClick.ts`) Плохая типизация в коде: отсутствие импорта как типа (строки 1, 3).
2. (`hooks/useOutsudeClick.ts`) Плохая логика в коде: Слишком длинное условие (строка 6).

3. (`components/dropdown.tsx`) Плохая типизация в коде:
    a. Отсутствие желательного `| null` в `useRef` (строка 7).
    b. Ошибка типов, `useOutsideClick` не принимает тип `null` (строка 9).
4. (`components/dropdown.tsx`) Плохая логика в коде:
    a. Отсутствуют параметры функции (пропсы) (строка 5).
    b. Отсутствие обработки нулевого значение `dropdownRef` (строки 7-9).
    c. Отсутствие возможности вставки дочерних компонентов.
    d. Лишний импорт `React` (строка 1).
5. (`components/modal.tsx`) Плохая типизация в коде: Отсутствие выноса типа параметров функции (строка 4).
6. (`components/modal.tsx`) Плохая логика в коде:
    a. Отсутствие желательного обратного условия (строка 6).
    b. Использование магических строк (строка 8).
    c. Отсутствие возможности использования дочерних элементов (строки 4, 25-28).
    d. Лишний импорт `React` и `useState` (строка 1).
7. (`App.tsx`) Плохая логика в коде: Лишний импорт `React` (строка 1).

Выводы:

Общее качество кода

На основе общего количества ошибок можно сделать вывод, что Алиса и DeepSeek показали сопоставимые результаты, в то время как ChatGPT допустил значительно больше ошибок.

Слабые места нейросетей:

Алиса — логические ошибки и некорректная работа с Ref в React.

DeepSeek — плохая типизация в React-компонентах и **сложная** для чтения **логики** (вложенные и длинные условия).

ChatGPT — Худший ассистент среди тройки. Он часто **переусложняет решение**, что приводит к большому количеству ошибок как в типизации, так и в логике.

Распределение ошибок по типам:

Плохая логика — самая распространенная категория ошибок (24 из 42), что указывает на то, что нейросети часто не могут выстроить корректную и простую последовательность действий.

Плохая типизация — вторая по распространенности проблема (16 из 42), особенно в контексте React, что критично для TypeScript-разработки.

В итоге:

Обобщенно говоря, все эти факторы говорят нам, что **нейросети** никак **не смогут заменить программистов**, по крайней мере в ближайшее время, однако они (нейросети) **будут мощным инструментом** для написания кода и замены написания рутинного кода.

Можно ли использовать нейросети в промышленной разработке?

Да и вот почему:

1. Нейросети являются мощным инструментом для написания кода.
2. Нейросети хорошо справляются с прототипированием и генерацией шаблонного кода.
3. Нейросети могут хорошо решать алгоритмические задачи.
4. Нейросети могут ускорить рутинные операции, такие как генерация простых методов, каркас класса и тому подобное.

Нейросети работают как "инициативный junior-разработчик." Модель может предложить несколько вариантов решения задачи, о которых можно

было не знать, а также они могут писать код быстрее, чем вы будете искать пример в документации.

Конкретные модели (Алиса, DeepSeek) показали себя достаточно стабильно для определенного класса задач (алгоритмы, утилиты), где их ошибки легко обнаружить и исправить.

Также вы можете дополнительно посмотреть исследование влияние ИИ на занятость: https://digitaleconomy.stanford.edu/wp-content/uploads/2025/08/Canaries_BrynjolfssonChandarChen.pdf.

Примечание: лучше всего использовать нейросети, сделанный под написание кода, такие как: Github Copilot, Claude, Qwen3 и подобные им. Мы их не использовали из-за того, что подобные нейросети требуют оплату за использование.

Список литературы:

1. Интернет-ресурс:
    1.1. Хабр “Нейронные сети: практическое их применение” — https://habr.com/ru/articles/322392 (дата последнего обращения: 26.10.2025).
    1.2. vc.ru “Нейросети в мире программирования: От текстов до кода” — https://vc.ru/services/858329-neiroseti-v-mire-programmirovaniya-ot-tekstov-do-koda (дата последнего обращения: 26.10.2025).
    1.3. Хабр “Ликбез по типизации в языках программирования” — https://habr.com/ru/articles/161205 (дата последнего обращения: 26.10.2025).
    1.4. Merion “Типизация языков программирование: что это и зачем она нужна” — https://wiki.merionet.ru/articles/tipizaciia-iazykov-programmirovaniia-cto-eto-i-zacem-ona-nuzna (дата последнего обращения: 26.10.2025).
    1.5. Википедия “Класс (программирование)” — https://ru.wikipedia.org/wiki/Класс_(программирование) (дата последнего обращения: 26.10.2025).
    1.6. MDN “Возвращаемые значения функций” — https://developer.mozilla.org/ru/docs/Learn_web_development/Core/Scripting/Return_values (дата последнего обращения: 26.10.2025).

1.7. MDN “Принятие решений в вашем коде — условные конструкции” — https://developer.mozilla.org/ru/docs/Learn_web_development/Core/Scripting/Conditionals (дата последнего обращения: 26.10.2025).

1.8. Хабр “Генерация документации с использование JSDoc” — https://habr.com/ru/articles/572968 (дата последнего обращени: 27.10.2025).